DS3 : Mots synchronisants

Ce sujet est constitué d'un seul problème adapté du sujet d'option informatique Centrale 2017 en s'appuyant sur un travail préalable de Nathaniel Carré. Les documents et la calculatrice sont interdits. Les encadrés tels que celui-ci émaillant le sujet sont des consignes à respecter absolument.

Définitions

On appelle *machine* un triplet (Q, Σ, δ) où Q est un ensemble fini et non vide dont les éléments sont appelés *états*, Σ est un alphabet et δ est une application de $Q \times \Sigma$ dans Q appelée *fonction de transition*. Une machine peut donc être considérée comme un automate fini déterministe et complet sans notion d'états initiaux ou finaux. On définit comme pour un automate la *fonction de transition étendue* $\delta^\star$ par :

- pour tout $q \in Q$, $\delta^\star(q, \varepsilon) = q$
- pour tous $q \in Q$, $u \in \Sigma^\star$ et $a \in \Sigma$, $\delta^\star(q, ua) = \delta(\delta^\star(q, u), a)$.

Un mot $u \in \Sigma^\star$ est dit *synchronisant* pour une machine (Q, Σ, δ) s'il existe $q_0 \in Q$ tel que pour tout $q \in Q$, $\delta^\star(q, u) = q_0$. L'existence de tels mots permet de ramener une machine dans un état spécifique en lisant un mot donné ; en pratique, cela permet de réinitialiser une machine réelle. Par exemple, les mots ba et bb dont des mots synchronisants pour la machine M_0 ci dessous.

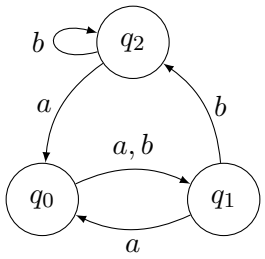

FIGURE 1 - La machine M_0.

Partie 1 *Considérations générales*

Les questions de programmation de cette partie seront traitées en utilisant la syntaxe du langage C. On supposera que les bibliothèques stdlib.h, stdbool.h et string.h ont été chargées.

On considère que l'alphabet Σ que l'on manipule dans le sujet est celui des lettres latines minuscules (de 'a' à 'z'). Les éléments de Σ seront donc représentées en C par des objets de type char et les mots de $\Sigma^\star$ par des objets de type char*. On rappelle que le code ASCII associé au caractère 'a' est 97.

Une machine est représentée à l'aide de la structure suivante :

```
struct machine{
    int taille_Q;
    int taille_Sigma;
    int** delta;
};
typedef struct machine machine;
```

Plus précisément, si $M = (Q, \Sigma, \delta)$ est une machine représentée par un objet m de type machine alors :

- m.taille_Q représente $|Q|$. On supposera toujours que les états de Q sont numérotés consécutivement à partir de 0, autrement dit que $Q = [\![0, |Q| - 1]\!]$.
- m.taille_Sigma représente $|\Sigma|$. On confondra une lettre et son numéro dans l'implémentation d'une machine ; étant entendu que le numéro du caractère 'a' est bien sûr 0.
- m.delta correspond à la table des transitions : si $q \in Q$ et $a \in \Sigma$ alors m.delta[q][a] vaut $\delta(q, a)$.

1. Ecrire une fonction machine* initialiser_machine(int tQ, int tSigma) renvoyant (un pointeur sur) une machine (Q, Σ, δ) telle que $|Q| =$ tQ, $|\Sigma| =$ tSigma et telle que pour tous $q \in Q$ et $a \in \Sigma$, $\delta(q, a) = q$.
2. Ecrire une fonction void liberer_machine(machine* m) libérant l'espace occupé par une machine.
3. Que peut-on dire de l'ensemble des mots synchronisants pour une machine ayant un seul état ?

Dans la suite du problème, on suppose que les machines considérés ont au moins deux états.

4. La figure 2 ci-dessous décrit une machine M_1 sur l'alphabet $\Sigma = \{a\}$. Donner un mot synchronisant pour M_1 s'il en existe un. Dans tous les cas, justifier la réponse.

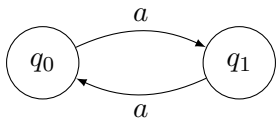

FIGURE 2 - La machine M_1.

5. Donner un mot synchronisant de trois lettres pour la machine M_2 décrite en figure 3.

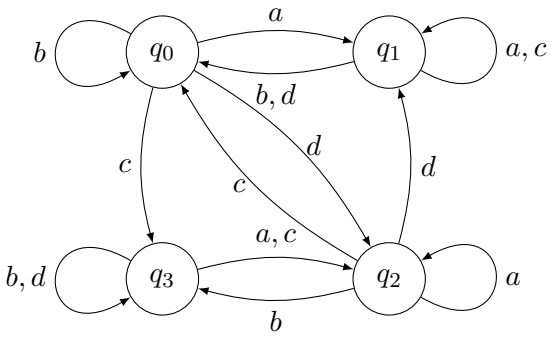

FIGURE 3 - La machine M_2.

6. Ecrire une fonction int delta_etoile(machine* m, int q, char* u) prenant en arguments une machine (Q, Σ, δ), un état $q \in Q$, un mot u et renvoyant l'état $\delta^\star(q, u)$.
7. Ecrire une fonction bool est_synchronisant(machine* m, char* u) renvoyant true si le mot u est synchronisant pour la machine m et false sinon.

Si $M = (Q, \Sigma, \delta)$ est une machine, on note $S(M)$ le langage formé par les mots synchronisants de M. La machine des parties associée à M est la machine $\overline{M} = (\overline{Q}, \Sigma, \overline{\delta})$ définie par $\overline{Q} = \mathcal{P}(Q)$ et

$$\forall\, P \subset Q, \forall\, a \in \Sigma,\ \overline{\delta}(P, a) = \{\delta(p, a) \,|\, p \in P\}$$

8. Montrer que l'existence d'un mot synchronisant pour M se ramène à un problème d'accessibilité de certain(s) état(s) depuis certains(s) état(s) dans $\overline{M}$.

9. En déduire que le langage des mots synchronisants de M est rationnel.
10. Déterminer et représenter graphiquement un automate fini et déterministe reconnaissant $S(M_0)$.

Partie 2 *Files*

> Les questions de programmation de cette partie et des suivantes seront traitées en utilisant la syntaxe du langage Ocaml. L'utilisation des fonctions des modules Array et List est autorisée. L'utilisation d'autres modules est interdite.

Dans cette partie, on cherche à implémenter une file d'attente à l'aide d'un tableau circulaire. On définit pour cela un type polymorphe file comme suit :

```
type 'a file = {donnees : 'a array ; mutable debut : int ; mutable taille : int}
```

Le champ debut indique l'indice du premier élément dans la file (donc, le premier qui en sortira). Le champ taille indique le nombre d'éléments dans la file. Les éléments de la file sont ceux dans le tableau donnees entre les indices debut et debut + taille -1 modulo la taille de donnees.

Par exemple, si on considère une file d'entiers dont le champ debut vaut 9, dont le champ taille vaut 5 et dont le champ donnees est représenté par le tableau suivant :

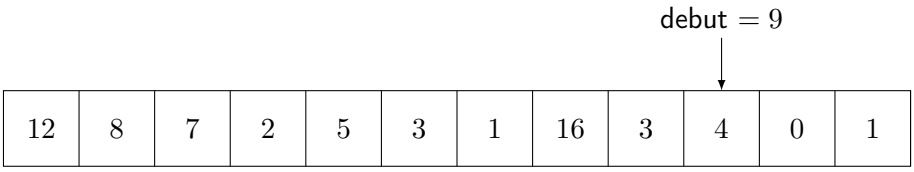

FIGURE 4 - Une file contenant $(4, 0, 1, 12, 8)$.

alors la file représentée est $(4, 0, 1, 12, 8)$, 4 étant sa tête.

11. Ecrire une fonction est_vide : 'a file -> bool et une fonction est_pleine : 'a file -> bool qui testent respectivement si la file en entrée est vide ou pleine et renvoient le booléen adéquat.
12. Ecrire une fonction enfiler : 'a file -> 'a -> unit prenant en argument une file et un élément x et ajoutant x à la file. S'il n'est pas possible d'enfiler, on affichera un message d'erreur explicite.
13. Ecrire une fonction defiler : 'a file -> 'a prenant en argument une file, modifiant cette file de sorte à en retirer la tête et renvoyant ladite tête. S'il n'est pas possible de défiler, on affichera un message d'erreur explicite.
14. Déterminer la complexité des fonctions de cette partie.

Partie 3 *Graphes d'automates*

On appelle *graphe d'automate* (sur un alphabet Σ) tout couple (S, A) où S est un ensemble fini de sommets et A une partie de $S \times \Sigma \times S$ dont les éléments sont appelés arcs. Si (q, a, q') est un arc, a est *l'étiquette* de l'arc, q son *origine* et q' son *extrémité*. Un graphe d'automate correspond ainsi au graphe sous-jacent à un automate potentiellement non déterministe et sans notion d'états initiaux ou finaux.

Par exemple, avec $\Sigma = \{a, b\}$, $S_0 = \{0, 1, 2, 3, 4, 5\}$ et $A_0 = \{(0, a, 1), (0, a, 3), (0, b, 0), (0, b, 2), (1, a, 1), (1, a, 2), (2, b, 1), (2, b, 3), (2, b, 4), (3, a, 2), (4, a, 1), (4, b, 5), (5, a, 1)\}$, le graphe d'automate $G_0 = (S_0, A_0)$ est représenté en figure 5 :

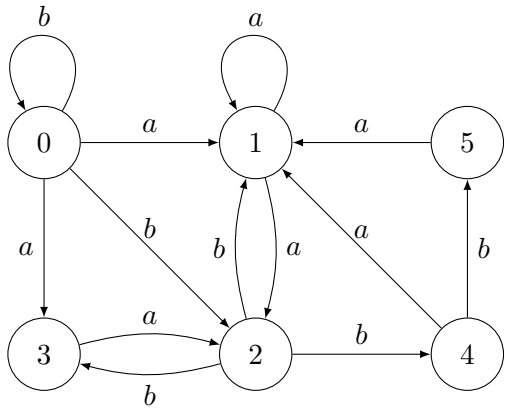

FIGURE 5 - Le graphe d'automate G_0.

Pour l'implémentation, on considère qu'un graphe d'automate aura toujours pour ensemble de sommets S l'intervalle d'entiers $[\![0, |S| - 1]\!]$ et pour alphabet Σ l'intervalle d'entiers $[\![0, |\Sigma| - 1]\!]$ (comme dans la partie 1, la lettre a sera ainsi représentée par l'entier 0). L'ensemble des arcs d'un tel graphe sera encodé par un tableau de liste d'adjacences : pour tout $s \in S$, la liste située à l'indice s dans ce tableau contiendra dans un ordre quelconque les couples (t, a) tels que $(s, a, t) \in A$.

Ainsi, le type suivant permet de manipuler des graphes d'automates :

```
type graphe = (int*int) list array
```

Par exemple, le graphe G_0 présenté en figure 5 sera représenté à l'aide de `g0` défini comme suit :

```
let g0 = [|[(1,0);(3,0);(0,1);(2,1)];
           [(1,0);(2,0)];
           [(1,1);(3,1);(4,1)];
           [(2,0)];
           [(1,0);(5,1)];
           [(1,0)]|]
```

Dans le pseudo-code ci-dessous, ∞, *vide* et *rien* sont des valeurs particulières qu'on spécifiera par la suite lorsqu'il s'agira d'implémenter l'algorithme correspondant :

Entrée : Un graphe d'automate $G = (S, A)$ et un ensemble de sommets $X \subset S$
Sortie : ?

```
1. mystère(G, X) =
2.     n ← |S|
3.     F ← file vide
4.     D ← tableau de taille n contenant ∞ dans chaque case
5.     P ← tableau de taille n contenant vide dans chaque case
6.     c ← n
7.     Pour tout s ∈ X
8.         Enfiler s dans F
9.         D[s] ← 0
10.        P[s] ← rien
11.        c ← c − 1
12.    Tant que F n'est pas vide
13.        s ← défiler F (au sens de la question 13)
14.        Pour tout (s, a, t) ∈ A tel que D[t] = ∞
15.            D[t] ← D[s] + 1
16.            P[t] ← (s, a)
17.            Enfiler t dans F
18.            c ← c − 1
19.    Renvoyer (c, D, P)
```

15. Justifier la terminaison de l'algorithme mystère.

16. Déterminer sa complexité en fonction de $|S|$ et $|A|$.

17. Justifier qu'au début de chaque passage dans la boucle tant que, si F contient les sommets $s_1, ..., s_r$ dans l'ordre dans lequel ils ont été ajoutés, alors $D[s_1] \leq D[s_2] \leq ... \leq D[s_r]$ et $D[s_r] - D[s_1] \leq 1$.

Pour tout sommet s de G, on note d_s la distance de X à s c'est-à-dire la longueur d'un plus court chemin possible entre un sommet de X et le sommet s (avec la convention $d_s = \infty$ s'il n'existe pas de tel chemin).

18. Justifier qu'à la fin de l'algorithme, pour tout sommet s, $D[s] \neq \infty$ si et seulement si s est accessible depuis un des sommets de X et que dans ce cas $d_s \leq D[s]$.

19. Montrer qu'en fin d'algorithme, on a en fait pour tout sommet s, $D[s] = d_s$.

20. En fin d'algorithme, à quoi correspondent c et $P[s]$ pour $s \in S$?

21. Ecrire une fonction `accessibles : graphe -> int list -> int * int array * (int*int) array` prenant en entrée un graphe d'automate G et un ensemble X de sommets de ce graphe (sous forme de liste) et renvoyant le triplet (c, D, P) calculé par l'algorithme mystère. Les constantes ∞, *vide* et *rien* seront respectivement codées par -1, $(-2, -1)$ et $(-1, -1)$.

22. Ecrire une fonction `chemin : int -> (int*int) array -> int list` prenant en entrée un sommet s et le tableau P calculé à l'aide de la fonction `accessibles` appliquée à G et X. Elle doit renvoyer un mot (sous forme de liste d'entiers) de longueur minimale qui est l'étiquette d'un chemin allant d'un sommet de X à s ou un message d'erreur s'il n'y en a pas.

Partie 4 *Existence d'un mot synchronisant*

Dans cette partie, on conçoit un algorithme permettant de déterminer si une machine $M = (Q, \Sigma, \delta)$ donnée admet un mot synchronisant. Si $X \subset Q$ et $u \in \Sigma^\star$, on note $\delta^\star(X, u)$ l'ensemble $\{\delta^\star(q, u) \,|\, q \in X\}$.

23. Soit u un mot synchronisant pour M et notons u_i le préfixe de longueur i de u pour tout $i \in [\![0, |u|]\!]$. Que peut-on dire de la suite des cardinaux $|\delta^\star(Q, u_i)|$?

24. Montrer que M admet un mot synchronisant si et seulement si pour tout couple d'états $q, q' \in Q$ il existe un mot $u_{q,q'}$ tel que $\delta^\star(q, u_{q,q'}) = \delta^\star(q', u_{q,q'})$.

Pour se servir du critère ci-dessus afin de répondre à la question qui nous occupe dans cette partie, on associe à la machine M la machine $\widetilde{M} = (\widetilde{Q}, \Sigma, \widetilde{\delta})$ définie par :

- $\widetilde{Q}$ est l'ensemble des parties de Q ayant un élément ou deux éléments.
- pour tout $X \in \widetilde{Q}$, pour tout $a \in \Sigma$, $\widetilde{\delta}(X, a) = \{\delta(q, a) \,|\, q \in X\}$.

25. Si $|Q| = n$, que vaut $\widetilde{n} = |\widetilde{Q}|$?

Du point de vue de la modélisation informatique, l'ensemble d'états d'une machine est un intervalle d'entiers consécutifs à partir de 0. Ainsi, on souhaite modéliser $\widetilde{Q}$ par $[\![0, \widetilde{n} - 1]\!]$. On note φ_n une bijection de $\widetilde{Q}$ dans $[\![0, \widetilde{n} - 1]\!]$.

26. En détaillant la bijection φ_n choisie, écrire une fonction `phi : int -> int list -> int` prenant en arguments l'entier n correspondant au nombre d'états de la machine M initiale, une liste à un ou deux éléments distincts ℓ représentant un état de $\widetilde{M}$ et renvoyant l'entier $\varphi_n(\ell)$.

On admettra disposer de sa fonction réciproque `phi_inv : int -> int -> int list`. Pour la suite, on représente une machine $M = (Q, \Sigma, \delta)$ par une matrice d'entiers `m` telle que `m.(q).(a)` représente l'état $\delta(q, a)$:

```
type machine = int array array
```

Par exemple, la machine M_0 dessinée en figure 1 correspond à :

```
let m0 = [| [|1; 1|]; [|0; 2|]; [|0; 2|] |]
```

27. Ecrire une fonction `delta_tilde : machine -> int -> int -> int` prenant en entrée une machine M, l'entier correspondant au numéro de l'état $X \in \widetilde{Q}$ et $a \in \Sigma$ et renvoyant (le numéro de) $\widetilde{\delta}(X, a)$.

On associe à la machine $\widetilde{M}$ un graphe d'automate $\widetilde{G}$ de manière transparente : l'ensemble des sommets de $\widetilde{G}$ est $\widetilde{Q}$ et son ensemble d'arcs est $\{(X, a, \widetilde{\delta}(X, a)) \,|\, (X, a) \in \widetilde{Q} \times \Sigma)\}$. On rappelle que si G est un graphe, alors son transposé, noté G^T est le graphe ayant les mêmes sommets que G mais dans lequel le sens de tous les arcs est inversé.

28. Ecrire une fonction `transpose : machine -> graphe` qui à partir d'une machine M calcule le graphe $\widetilde{G}^T$ où $\widetilde{G}$ est le graphe d'automate associé à $\widetilde{M}$.

29. Justifier qu'il suffit d'appliquer la fonction `accessibles` de la partie 3 au couple $(\widetilde{G}^T, X)$ avec X l'ensemble des sommets de $\widetilde{G}^T$ correspondant à des singletons pour déterminer si la machine M possède un mot synchronisant.

30. Ecrire une fonction `existe_synchronisant : machine -> bool` qui détermine si une machine admet un mot synchronisant.

Partie 5 *Réduction depuis SAT*

La partie précédente montre qu'on peut déterminer algorithmiquement si une machine admet un mot synchronisant. En pratique, on souhaite trouver un tel mot de taille la plus petite possible. En effet, plus ce mot est court, moins d'opérations sont nécessaires pour réinitialiser une machine réelle (donc moins les pièces de la machine s'usent à faire des actions à vide et moins de temps est nécessaire à la réinitialisation).

Dans cette partie, on appelle "Mot synchronisant" le problème de décision défini par :

> **Entrée :** Une machine M et un entier m.
> **Sortie :** Oui si M possède un mot synchronisant u tel que $|u| \leq m$; non sinon.

On va montrer que ce problème est au moins aussi difficile que le problème CNF-SAT défini par :

> **Entrée :** Une formule φ du calcul propositionnel en forme normale conjonctive.
> **Sortie :** Oui si φ est satisfiable ; non sinon.

Pour ce faire, pour chaque instance φ de CNF-SAT, on va construire une machine M_φ telle que déterminer la satisfiabilité de φ revient à résoudre le problème du mot synchronisant pour une longueur m donnée pour M_φ. Soit φ une formule sous forme normale conjonctive composée de n clauses $C_1, ..., C_n$ et faisant intervenir m variables $x_1, ..., x_m$. On définit $M_\varphi = (Q, \Sigma, \delta)$ par :

- Q est formé de $mn + n + 1$ états : un état particulier noté f et $n(m+1)$ autres états qu'on notera $q_{i,j}$ pour tous $(i,j) \in [\![1, n]\!] \times [\![1, m+1]\!]$.
- $\Sigma = \{0, 1\}$.
- δ est définie par :

  $\delta(f, 0) = \delta(f, 1) = f$ (autrement dit, f est un état puits),

  pour tout $i \in [\![1, n]\!]$, $\delta(q_{i,m+1}, 0) = \delta(q_{i,m+1}, 1) = f$,

  pour tout $i \in [\![1, n]\!]$ et tout $j \in [\![1, m]\!]$,

$$\delta(q_{i,j}, 0) = \begin{cases} f \text{ si le littéral } \overline{x_j} \text{ apparaît dans la clause } C_i \\ q_{i,j+1} \text{ sinon} \end{cases}$$

$$\delta(q_{i,j}, 1) = \begin{cases} f \text{ si le littéral } x_j \text{ apparaît dans la clause } C_i \\ q_{i,j+1} \text{ sinon} \end{cases}$$

31. On considère la formule φ_0 à 3 clauses et 4 variables suivantes :

$$\varphi_0 = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee x_4)$$

    Représenter graphiquement la machine M_{φ_0} en omettant les transitions menant vers l'état puits.

32. Donner une valuation v satisfaisant φ_0. Le mot $v(x_1)v(x_2)v(x_3)v(x_4)$ est-il synchronisant pour M_{φ_0} ?

33. Montrer que tout mot de longueur $m+1$ est synchronisant pour M_φ. A quelle condition sur les $\delta^\star(q_{i,1}, u)$ un mot u de longueur m est-il synchronisant pour M_φ ?

34. Montrer que si φ est une formule sous forme normale conjonctive satisfiable, alors tout modèle de φ fournit un mot de longueur m synchronisant pour M_φ. On détaillera la construction de ce mot.

35. Montrer réciproquement que si M_φ possède un mot synchronisant de longueur inférieure ou égale à m alors φ est satisfiable. Décrire comment construire une valuation pour φ dans ce cas.